Parallelization of the Traffic Flow Management Problem

Joseph Rios *
NASA Ames Research Center, Moffett Field, CA 94035
Kevin Ross†
University of California at Santa Cruz, Santa Cruz, CA 95064

This paper describes a parallel approach to solving the national air traffic flow management problem. High fidelity, aircraft-level approaches to this problem can determine which flights should be held on the ground or in an enroute sector. The major drawback to such aircraft-by-aircraft approaches lies in the large amount of data and resulting massive problem instances to be solved for nationwide or long planning horizon scenarios. The presented approach solves twenty individual problems, one for each Air Route Traffic Control Center in the continental United States. Individual client processes coordinate with a central server to solve their respective problems over multiple iterations. Experiments are performed using recent, historic data within a nominal scenario and a weather scenario. These experiments demonstrate the potential for solving this problem using parallel approaches in greatly reduced time. The initial results show solutions that would have taken over 24 hours to obtain optimally in a monolithic system were shown obtainable to within 3% optimality in less than one hour using the parallel architecture. In addition to the runtime and delay cost analysis, delay results from successfully running a high-fidelity, nationwide traffic flow scenario are also detailed.

I. Introduction

Given a set of flights and a set of sector and airport capacity constraints in the National Airspace System (NAS), traffic flow management (TFM) attempts to determine how flights should be scheduled (or re-scheduled) to satisfy the constraints.[1] Models that solve the problem with a high level of detail are computationally intensive due to a large solution space. The models that offer the highest level of detail are those which treat each flight discretely. There are several examples of such aircraft-by-aircraft level models for the TFM problem (TFMP) and related problems.[1,2] Other models attempt to capture the dynamics of the system through aggregation of flights.[3,4] The main reason for aggregating flights into flows is to make the problem computationally tractable. One issue with aggregation methods is the translation of the solution into an action plan. This implies that the aggregated flights will need to be disaggregated to decide which flights need to be delayed and where that delay needs to occur. A solution from a discrete model is theoretically ready for implementation as a TFM action. Thus, to be useful, an aircraft-level model would need to be solved in a reasonable amount of time for the problem at hand. Currently, high-fidelity TFMP instances of any significant size cannot be solved within a time window to make the solution relevant for a live problem. For example it may take well over an hour to solve a 1 hour planning horizon problem. To date, the authors have not found work that specifically addresses the acceleration of solving a discrete TFMP model.

Research into the acceleration of algorithms can branch in several directions. New hardware can be developed to perform computationally intensive tasks,[5,6] other methods may focus on algorithm optimization.[7] This paper investigates still another common method by trying to break free from the serial nature of modern CPUs and parallelize some portion of the algorithm such that it may be simultaneously worked

*Aerospace Engineer, Automation Concepts Research Branch, Mail Stop 210-10, Joseph.L.Rios@nasa.gov, 650-604-0231.
†Assistant Professor, School of Engineering, 1156 High Street, kross@soe.ucsc.edu, 831-459-1878.

upon by several CPUs or processing threads at once. There is a large body of work on solving large-scale optimization problems with parallelization schemes.[8] Most research on the parallelization of optimization focuses on general techniques and not as much on domain-specific problems.

The goal of this paper is to demonstrate that a large, high-fidelity, discrete TFMP instance can be solved in significantly less time as a parallel problem than it can as a traditional, serial problem. To this end, solution quality is a primary concern. While obtaining a globally optimal solution is generally always possible in a monolithic instance of a TFMP, obtaining that same solution may be quite difficult for a parallel version. Thus, while parallelizing the problem, the solution quality is monitored to verify that the parallel solution is not significantly worse (i.e., less than 5%) than the monolithic solution. The core contributions of this paper can be summarized as follows:

1. The TFMP is cast as a parallel problem solvable by modern multi-core computers and/or a network of computers.

2. Runtime improvements and solution quality are measured to determine the potential value of a parallel approach to the TFMP.

3. Results from applying a discrete model to the TFMP to nationwide scenarios are analyzed.

The remainder of the paper is organized as follows. In section II the model for describing the TFMP is presented. Then in section III, the parallel version of the model is described in detail. Next, in section IV the input data and its obtainment for our experiments is detailed. In section V the experiments are detailed with results and analysis provided. Finally, in section VI a conclusion is presented along with future research opportunities.

II. The Bertsimas Model of the TFMP

The model presented by Bertsimas and Stock-Patterson[1] (BSP) is a well-understood and often used[9,10] optimal approach to the aircraft-level TFMP. This model solves the TFMP with multiple airports and deterministic sector capacities. Each flight in the set of flights, $\mathcal{F}$, is described as an ordered list of sectors with earliest and latest feasible entry times for each of those sectors. Sectors in the flight path are denoted by $P(f, y)$, where f is the flight and y is the ordinal representing the sector in the flight path. For the purposes of the model, airports are considered as special cases of sectors. The objective function is of the form:

$$\text{Min} \sum_{f} [c_f^g g_f + c_f^a a_f]$$

where the c_f^g and c_f^a are the costs of holding a flight on the ground or in the air, respectively, for one unit of time. Generally, air holding is considered more expensive than ground holding due to fuel costs, so an air cost (c_f^a) to ground cost (c_f^g) ratio of 2:1 is used througout this work. This encourages flights to be held on the ground, but allows for air holding when necessary. The air and ground delay for each flight f (a_f and g_f, respectively) are ultimately expressed in terms of the binary variables, w, through a substitution for a_f and g_f. These variables designate whether a flight has entered a given sector by a given time. The substitution involves determining when the flight actually departed and actually arrived versus the times it was scheduled to do so.

$$w_{ft}^j = \begin{cases} 1, & \text{if flight } f \text{ arrives at sector } j \text{ by time } t, \\ 0, & \text{otherwise.} \end{cases}$$

The problem is constrained by various capacity restrictions. Namely, airport departure ($D_k(t)$), airport arrival ($A_k(t)$), and sector ($S_j(t)$) capacities at time t, where k is in the set of airports, $\mathcal{K}$, and j is in the set of sectors, $\mathcal{J}$:

$$\begin{array}{rcl} \sum_{f:P(f,1)=k} (w_{ft}^k - w_{f,t-1}^k) & \leq & D_k(t) \\ \sum_{f:P(f,\text{last})=k} (w_{ft}^k - w_{f,t-1}^k) & \leq & A_k(t) \\ \sum_{f:P(f,i)=j,P(f,i+1)=j'} (w_{ft}^j - w_{ft}^{j'}) & \leq & S_j(t) \end{array}$$

There is a set of constraints that guarantees each flight spends at least the specified minimum amount of time in each of its sectors. A final set of constraints enforces the flight path requirements, i.e., the sectors are visited in the correct order. These two constraint sets are formalized here, respectively:

$$\begin{aligned} w^{j'}_{f,t+\min(f,j)} - w^{j}_{ft} &\leq 0 \\ w^{j}_{ft} - w^{j}_{f,t-1} &\geq 0 \end{aligned}$$

The model, as originally described, allows for continuing flights (i.e., use of the same plane on more than one flight). Continuing flights were ignored here, so all of the flights in this study are assumed to be 'single-leg.' Continuing flights are easily accommodated abstractly in the model, though are often difficult to obtain in real data sets. The original authors note how this model is also extensible to various scenarios including rerouting of flights, modeling of ground-holding programs, modeling of banks of flights wherein one of several planes may be assigned to a given flight, and modeling specific airports that exhibit dependence between arrival and departure capacities.

III. Parallel Architecture

To solve the TFMP the architecture described here divides the data for the problem into n subproblems. The general idea will be to allow each subproblem to be solved by a separate solver. Over multiple iterations, solutions from the individual solvers are coordinated by a central server. Solutions are reported to this central server so that it may synthesize and/or analyze the individual solutions with the goal of finding a feasible, global solution. Obtaining this global solution involves sending updates to the individual solvers between iterations so that each local solution is eventually consistent with every other solver's view of the data. The algorithm is defined precisely below in Algorithm 1 and described in more detail afterward.

Algorithm 1 Solving the TFMP in parallel

```
Run simulation
Generate data files
firstIteration ← true
Launch individual solvers
for each solver in parallel do
    Establish contact with server
    Await "go solve" message from server
end for
repeat
    Send "go solve" message to all solvers
    for each solver in parallel do
        if firstIteration then
            firstIteration ← false
        else
            Update data based on messages received
        end if
        Solve local problem
        Report delayed flights to server
        Report local objective value to server
    end for
    Send updated data messages to appropriate solvers
until stopping criteria met
Send "finished" message to all solvers
Report global results
```

The architecture implemented can be described as *globally synchronized*.[11] This means that each solver solves its individual problem, reports its results to a central server and then waits until further instructions are received from the server. The server waits until all individual solvers complete before issuing any potential data updates and a subsequent signal to all solvers to solve again or quit.

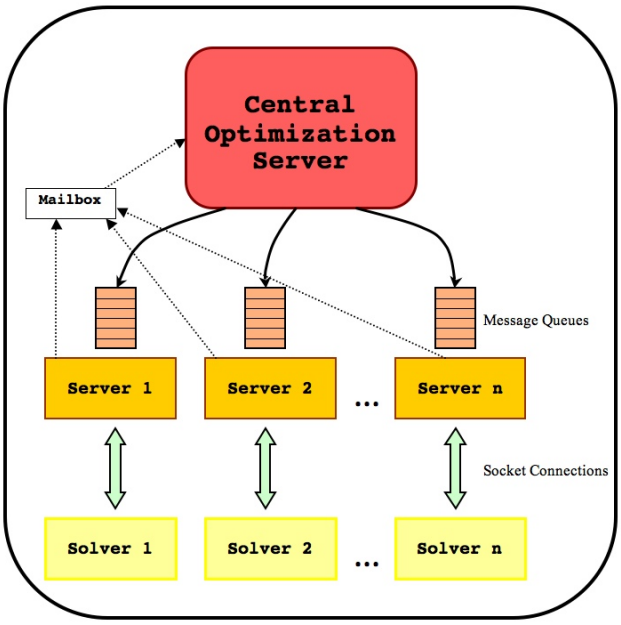


Figure 1. The high-level architecture of the parallel solver.

The central server generates data update messages as each solver reports its results. The results consist of a series of character strings describing either the holding of any delayed flights or the solver's total local delay. Other solvers are made aware of any delays that affect flights within its subproblem. For example, if DAL100 is tagged for 5 minutes of ground delay by the solver controlling DAL100's departure airport, all solvers handling the subsequent lengths of DAL100's flight path need to be made aware of this delay, but no other solvers need be notified. If a solver is sent a message before it completes its current solve cycle, the message is buffered until the solver is ready to process it. After all solvers report their results and the central server sends out all data update messages, a broadcast message is sent to all solvers to either solve again or quit, depending upon the central server's stopping criteria.

There are three stopping criteria wherein if any one of the criteria are met, the system will stop. First, if the total delay in the system is found to be zero, there is no need to solve again, as zero is an absolute minimum solution. The second criterion involves checking for changes between iterations. If no solver reported any change in delays between iterations, the server could safely assume a feasible solution had been reached, and there would be no need to solve again. Finally, there is a parameter to cause the system to stop after a set number of iterations. In general, the more subproblems that are being used, the larger this parameter should be. For instance, all experiments presented here required at most 3 iterations before halting due to the second criterion noted above. Essentially, with more subproblems, there will be a greater need to coordinate solutions between iterations.

The central server can be modified to include any amount of logic to determine what messages are appropriate to send to the solvers between iterations. In the current implementation, the logic is simple, yet effective. Essentially, for any delay request by an individual solver, the central server enforces that delay request upon all involved solvers. To continue with the example of flight DAL100 above, after the first solver notifies the central server of the 5 minutes of ground delay, all other involved solvers are forced to honor *at least* that amount of delay. So if DAL100 is scheduled to fly through regions controlled by two other solvers, both of those solvers would modify their data to expect DAL100 to enter their respective airspace at least 5 minutes later than scheduled. In the future, this logic could be modified to make the 5 minutes of delay more of a request than a command, thus if delaying DAL100 5 minutes in New York causes a ripple effect delaying 3 other flights, that request could be rejected and ultimately reformulated by the requesting solver.

The central server initiates the solve process by launching the individual solvers. The solvers then contact the central server and are assigned a server listener with which to communicate. As mentioned earlier, messages are sent from the central server to message queues for the solvers and the solvers are able to communicate their results via a single "mailbox" owned by the central server. The high-level architecture is illustrated in Figure 1. Even though these experiments were performed on a multi-core machine, the solvers could be launched on various machines throughout a network and not affect the overall architecture of the system.

IV. Input Data

For the sake of consistency, a single day of historical data was used for all experiments. The chosen date was Thursday, August 24th, 2005. The data were extracted from the historical ASDI (Aircraft Situation Display to Industry)[12] files using NASA's Future ATM Concepts Evaluation Tool (FACET).[13] FACET was then used to filter flights. This project was focused on only domestic flights so all international traffic was excluded. Also, all flights that left a center and then re-entered the same center later without landing first were excluded; flights in this class were very few. Flights through Canadian airspace were kept, but the

Canadian sector constraints were ignored. Overall, after these exclusions roughly 90% of all traffic was kept. More specifically, the shortest planning horizon (20 minutes) described in the next section had 3845 flights under control of the model, while the longest planning horizon (160 minutes) had 9257 flights included.

After filtering, the remaining flights were simulated for various time horizons depending upon the particular experiment. All scenarios were started at 13:15 UTC (9:15 AM EDT) in order to capture a large volume of traffic that covered most of the contiguous United States. An earlier start time would have left minimal traffic on the west coast, and any later start time would have missed much of the 'rush' on the east coast. To generate data files for the specific experiments, the pre-filtered data file described in the previous paragraph was run though FACET as a simulation. Information about each flight was obtained through the FACET Application Programming Interface (API). The FACET API is relatively new and based upon the CARAT# (Configurable Airspace Research and Analysis Tool-Scriptable) tool.[14] Updated information about the NAS was recorded every minute of simulated time. The collected information contained the following for each flight:

1. Origin and destination airports.
2. Departure and arrival times.
3. Sectors along the flight's path.
4. Sector occupancy times.

The data obtained through this simulation are assumed to encapsulate the ideal intent of each flight. It is understood that, in reality, the schedules of these flights were likely not ideal for the airlines. They may have already been delayed due to some traffic flow management action, congestion, mechanical problems, or any other reason.

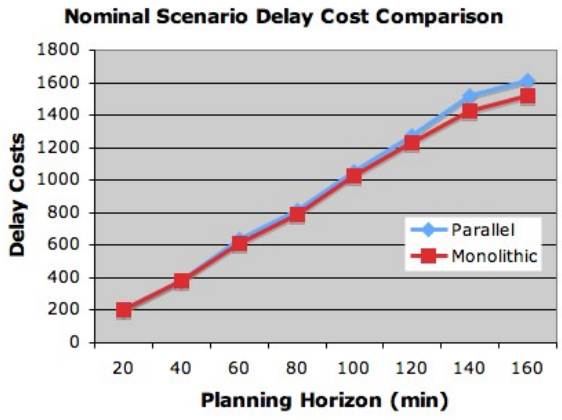
The maximum number of flights that should be within sector boundaries is represented by the Monitor Alert Parameter (MAP) value for that sector. For airports, there are capacities for arrivals and departures that are dictated by runway configuration and/or weather. The theoretical maximums for sector and airport capacities as understood by FACET were used as nominal capacity values.

V. Experiments and Results

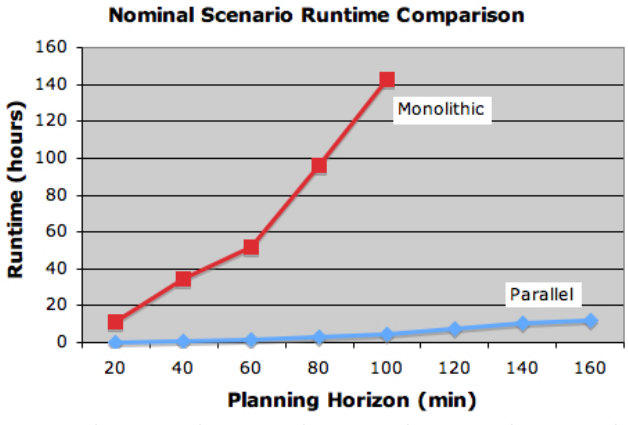
The twenty Air Route Traffic Control Centers ("Centers") in the continental U.S. served as the partitioning for the individual solvers in these experiments. Thus, n is equal to 20 in Figure 1. While this will not partition the problem into balanced subproblems, it was easiest to implement to prove the concept. A scheme for creating n balanced subproblems (e.g., n subproblems with roughly the same number of flights) is for future research. As an example of this imbalance, in the smallest experiment described below, the data files for the Seattle Center (ZSE) and the Atlanta Center (ZTL) were the smallest and largest, respectively, of all the Centers. Solving the ZSE subproblem takes roughly 8 seconds, while solving the ZTL subproblem takes roughly 240 seconds. The positive results presented below with these imbalanced subproblems highlight the promise of the approach.

The two main measurements of the parallel system will be of its runtime and solution quality as compared to an equivalent monolithic (unparallelized) version. To this end, scenarios based on the August 24th, 2005 data file were constructed. The first scenario looks at performance as the planning horizon increases from 20 minutes out to 160 minutes under "nominal" (default capacity/demand) conditions. The second set models a weather scenario wherein several Cleveland center sectors are impacted along with a foggy morning at San Francisco International Airport (SFO). The following subsections describe each set of experiments in detail and provide the results along with some analysis.

All experiments were executed on a dual 64-bit, 2.66 GHz Xeon quad-core processor system with 32GB of RAM. Thus, there were 8 individual cores available on which to run the individual solvers, and available memory was not a limiting factor. To solve these problems, there is a model generation phase followed by an actually solving phase. All time measurements are the sum of both of these times. The solver software was the GNU Linear Programming Kit (GLPK).[15] GLPK was chosen over the top-of-the-line, industry standard CPLEX[16] solver due to the need to run several instances of the solver simultaneously. Since GLPK is open source software and freely licensed, there was freedom to run these experiments with multiple, simultaneous instances as desired, while multiple licenses of CPLEX would have been necessary to perform the same work. There is a steep performance difference between GLPK and CPLEX on large problems. Sometimes this difference is two orders of magnitude and almost always at least one. For example, some experiments that



(a) The delay cost comparison between parallel and monolithic versions.



(b) The runtime comparison between parallel and monolithic versions.

Figure 2. Experimental results from increasing the planning time horizon in the nominal scenario. Note that the difference in delay costs remains relatively constant and small while the runtime dramatically diverges.

would not complete running in under a week with GLPK finished in about 200 minutes with CPLEX. As such, the presented results are valuable for comparison of the parallel version versus the monolithic version and not necessarily as an absolute measure of industry standard solver speed. A study that is beyond the scope of this work to verify is that the relative performance of the parallel architecture versus the monolithic version will remain constant despite the choice of solver software.

V.A. Increasing Planning Horizon

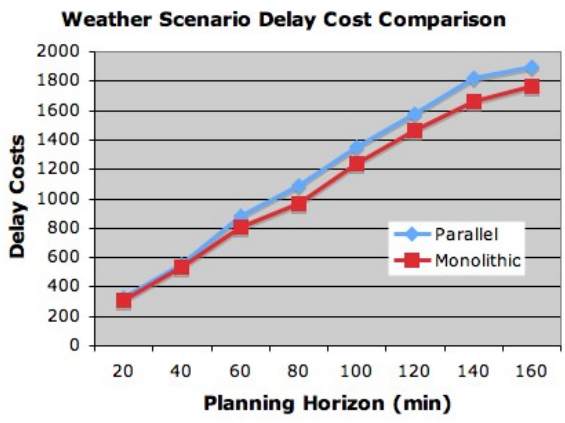
Using the data described in Section IV, experiments determined the performance of the parallel architecture as the TFM planning horizon steadily increased. Using 20-minute steps, the planning horizon was increased from 20 minutes up to 160 minutes, resulting in eight separate runs for each architecture. For each planning horizon, the solution quality (overall delay costs) and the runtime were compared between the parallel architecture and the monolithic version of the same scenario. Figure 2 illustrates the results. Note that GLPK was unable to solve the 120+ minute monolithic problems in less than a week's runtime, thus the optimal solutions for those scenarios were found using the more powerful AMPL[17]/CPLEX[16] software. This was necessary to compare a known optimal value to the solution obtained by the parallel architecture. Though the runtime results are not comparable and not shown on the graph.

The results indicate promising speedups for the parallel architecture at the small cost of increased relative delay compared to the monolithic global optimum. To achieve speedups of 20-30 times, the cost is only an average of 3% increase in delay over the global optimum.

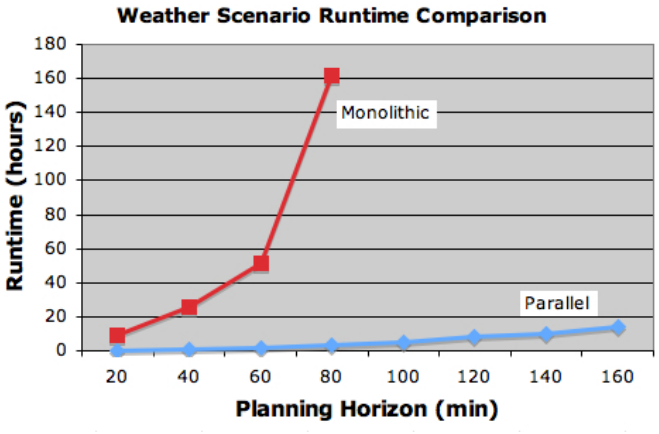
V.B. Weather Impacted Scenarios

To model the impact of weather on the NAS, various capacities in the system can be reduced. These reductions in capacities approximate the effects of flights avoiding certain sectors or delaying arrivals or departures due to airport conditions. Specifically for this experiment, three high-altitude, enroute sectors in the busy New York-Chicago corridor were reduced to 40% of their nominal capacities (MAP values), while arrivals to San Francisco International Airport (SFO) were reduced from a rate of 60 per hour to a rate of 32 per hour. The enroute sectors chosen were all contiguous and within the Cleveland Center (ZOB26, ZOB49 and ZOB69). To get a sense of the sectors' locations relative to Chicago's O'Hare International Airport (ORD) and Newark International Airport (EWR), Figure 4 is provided. The same planning horizons that were used in the nominal capacity scenario were also used here.

Results for the weather-impacted experiments are illustrated in Figure 3. Compared with the nominal scenario results, it is clear that the runtime improvements were consistent (i.e., large gap in performance between parallel and monolithic versions), but the quality of solution suffered slightly. The implication of this small set of data is that runtime improvements are likely guaranteed from within the parallel architecture,



(a) The delay cost comparison between parallel and monolithic versions in the weather scenario.



(b) The runtime comparison between parallel and monolithic versions in the weather scenario.

Figure 3. Experimental results from increasing the planning time horizon in the weather scenario.

while denser scenarios may yield some extra degradation in solution quality. Further experiments would need to be run to determine if these observations hold.

V.C. Analysis of Nationwide Delays

Since there have been no studies published describing the application of a nationwide, discrete, high-fidelity model to solve the TFMP, further details regarding the experiments in this project are now provided. The following data are from the weather scenario experiments described in Section V.B, as the results from the nominal scenario provide similar insight. The data provided describe the type of delay (air vs. ground) over the planning horizons, the amount of responsibility attributable to each center for nationwide delay and, finally, the amount of delay assigned to each delayed flight.

Figure 5 shows the breakdown of air vs. ground delay minutes as discovered by the two model implementations. The fundamental insight here is that air delay does not increase significantly from the 80 minute planning horizon to the 160 minute planning horizon in either implementation. When the delay values are scrutinized on a plane-by-plane basis, it becomes evident that nearly all air delay is due to flights already in the air at the beginning of the planning horizon. Thus these flights cannot be ground held in the presence of any demand/capacity imbalance along their respective flight paths. Nearly every flight that has not departed prior to the beginning of the planning horizon will take ground delay instead of air de-

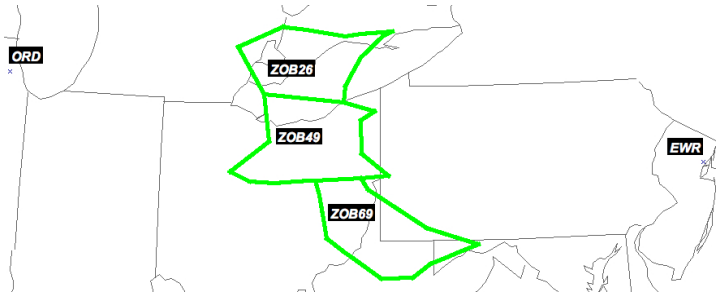


Figure 4. The sectors modeled with reduced capacity in the weather scenario experiments (ZOB26, ZOB49, ZOB69) are highlighted in green. State boundaries are visible in the background.

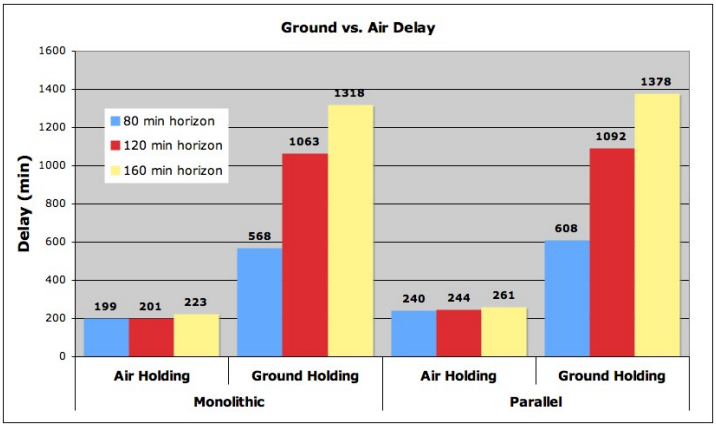


Figure 5. Breakdown of air delay and ground delay in the weather scenario for three planning horizons.

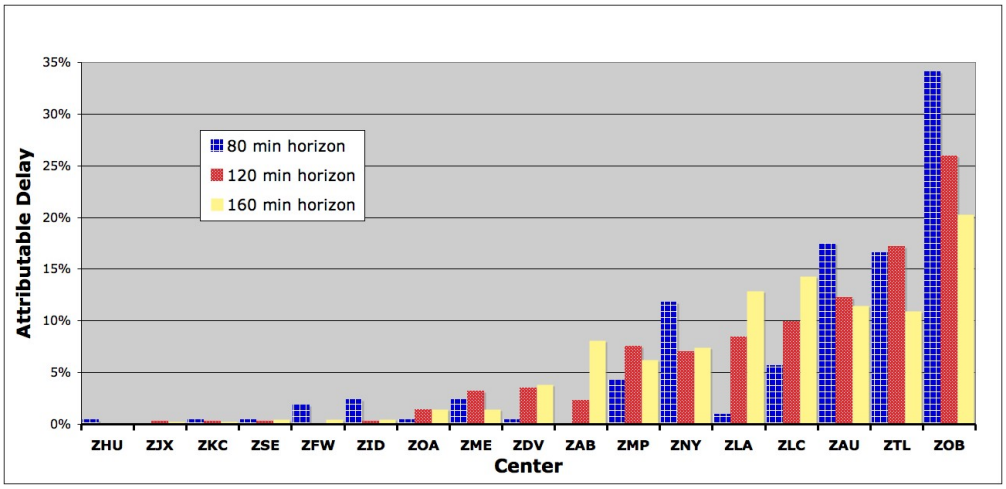


Figure 6. Amount of attributed delay by Center for the weather scenario. ZOB was modeled as having weather problems, thus most delay was attributable to that Center. Centers with zero attributable delay (ZMA, ZDC and ZBW) have been omitted.

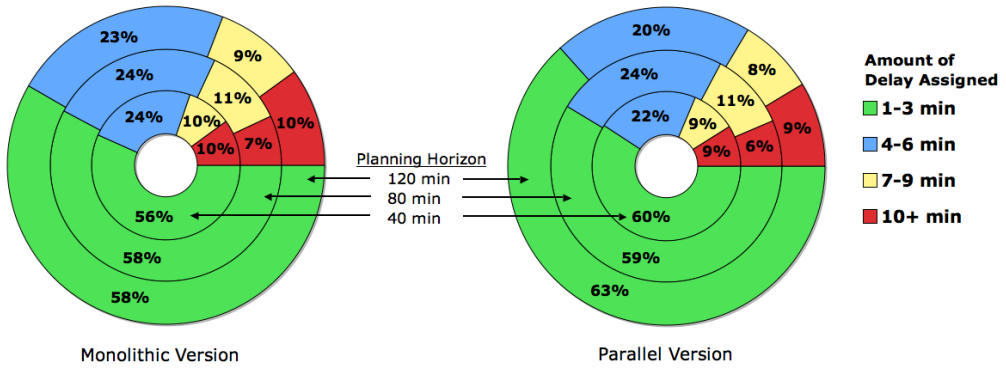


Figure 7. A breakdown of the amount of delay per delayed flight from the weather scenario. The majority of flights were delayed less than 4 minutes in both versions over all planning horizons.

lay if the need for delaying that particular flight arises. Ultimately, this is a function of the cost ratio of air holding (c_a) to ground holding (c_g) discussed in Section II. If this ratio were 1, then the expectation might be that there would be more air holding (but perhaps less overall delay), but such experiments were not part of this study.

Figure 6 shows the responsibility for delay by Center for the parallel version. A Center is deemed "responsible" for the delay if the final request for delay for a given flight came from that Center. Since the experiment assigned a solver to each Center, this distribution is easily tracked in the parallel version, but not so in the monolithic version. Thus, only the parallel version of these data is presented. Since these data are from the weather scenario, the fact that most of the delay is attributable to ZOB is not surprising, since that Center was modeled as having sectors with reduced capacity. The other Centers with large amounts of responsibility are those that are well-known for their heavy traffic loads. Also of note is the small amount of delay attributed to ZOA. This implies that the extra arrival constraint put on SFO did not impact the NAS significantly for this scenario and set of flights. This may have been expected given the time of scenarios and the exclusion of international traffic from the scenarios.

The final set of data depicts how the delay was distributed between the delayed flights. The donut charts in Figure 7 depict a breakdown of the amount of delay assigned to the delayed flights in the system for both the monolithic and parallel versions. The vast majority of flights were not delayed, so including all of the zero-delay flights would skew these results significantly. The most interesting detail of these data is the relatively "fair" distribution of delay. The majority of delayed flights were delayed less than 4 minutes in both scenarios for all planning horizons. While this equity will not necessarily hold for an arbitrary

scenario, it may indicate that, in general, solutions arising from the BSP model will be equitable for large enough scenarios such as those over an hour. All of this depends on several factors, not the least of which is one's definition of equity and further experimentation would be required to make any deeper statements regarding fairness. A further point on equity is the comparison of equity in charts of Figure 7. Based on this small set of experiments, it seems that the parallel version is slightly more "fair" in that more flights are given lower levels of delay. A promising future study may investigate whether this is a by-product of the parallel architecture itself. Perhaps since each solver has smaller flight lengths upon which to hold flights as compared to a monolithic version, shorter holds are applied to individual flights.

VI. Conclusions

The system presented solves large-scale, high fidelity TFM problems in a parallel, near-optimal fashion in significantly less time than a traditional, monolithic optimal approach. The model used within this parallel architecture considers each flight discretely, thus the obtained solutions are ready for implementation as a traffic management action plan. The parallel system was successfully implemented and measured using a state-of-the-art multi-core personal computer. The results presented in the previous section indicate a substantial decrease in computation time with small increase in overall delay costs. A separate analysis would be required to verify that the tradeoff is indeed acceptable, but moving from virtual intractability to solvability is likely worth the 1-6% increase in overall delay costs. Solutions that would have taken over 24 hours to obtain optimally were shown obtainable to within 3% optimality in less than one hour. More specifically, a scenario with 4632 flights was solved within 45 minutes on the parallel architecture, while the monolithic version of the same scenario took over 34 hours to solve. This particular 46.5X speedup came at a 1% increase in delay costs. As Figures 2 and 3 illustrate, an increasing planning horizon creates a large divergence between the runtimes of the monolithic and parallel systems, while the difference solution quality remains relatively constant. This implies that as larger, longer scenarios are attempted with the parallel architecture, some confidence in the solution quality can be assumed, though no bounds or guarantees are offered in this study. If state-of-the-art optimization tools could be used within this framework (instead of the open source tools used), solutions to large-scale, high-fidelity TFM problems could likely be obtained in near real-time, i.e., near optimal solutions could be guaranteed within some pre-set time limit. Before this study, there was no known system to attempt such a goal.

VII. Future Work

Future studies would likely consider several extensions or modifications. One of the first questions to be answered in the future would be how to better balance the individual sub-problems such that they could complete in roughly the same amount of time. Breaking the problem based solely on Center boundaries is not ideal, but did offer a good starting place for this study. Additionally, it might important to establish some mathematical bounds on the parallel system's performance so that confidence could be established in results produced. Another extension could involve eliminating the central server architecture and allowing the individual solvers to communicate directly, though it was not clear in this study that communication was a bottleneck. It would be informative to also see how well this parallelization approach might combine with other computation-time reduction methods[18] for this problem. Finally, it will be important to see how a system such as this would perform in a "live" scenario wherein updates to available information (flight plans, plane locations, weather systems, etc) are incorporated between runs.

References

[1]Bertsimas, D. and Patterson, S. S., "The Air Traffic Flow Management Problem with Enroute Capacities," *Operations Research*, Vol. 46, No. 3, May-June 1998, pp. 406–422.

[2]Bayen, A. M., Grieder, P., Meyer, G., and Tomlin, C. J., "Lagrangian Delay Predictive Model for Sector-Based Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 5, 2005, pp. 1015–1026.

[3]Sridhar, B., Soni, T., Sheth, K., and Chatterji, G., "An Aggregate Flow Model for Air Traffic Management," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, August 2004.

[4]Menon, P., Sweridul, G., and Bilimoria, K., "New Approach for Modeling, Analysis, and Control of Air Traffic Flow," *Journal of Guidance, Control, and Dynamics*, Vol. 27, No. 5, 2004, pp. 737–744.

[5]Schutz, B.; Klindworth, A., "A VLSI-chip for a hardware-accelerator for the simplex-method," *ASIC Conference and*

Exhibit, 1992., Proceedings of Fifth Annual IEEE International, 21-25 Sep 1992, pp. 553–556.

[6]Bayliss, S., Bouganis, C.-S., Constantinides, G. A., and Luk, W., "An FPGA implementation of the simplex algorithm," *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, Dec. 2006, pp. 49–56.

[7]Huang, J.C.; Leng, T., "Generalized loop-unrolling: a method for program speedup," *Application-Specific Systems and Software Engineering and Technology, 1999. ASSET '99. Proceedings. 1999 IEEE Symposium on*, 1999, pp. 244–248.

[8]Averick, B. M. and More, J. J., "Evaluation of Large-Scale Optimization Problems on Vector and Parallel Architectures," *SIAM Journal of Optimization*, Vol. 4, No. 4, November 1994, pp. 708–721.

[9]Rios, J. and Ross, K., "Delay Optimization for Airspace Capacity Management with Runtime and Equity Considerations," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Hilton Head, South Carolina, August 2007.

[10]Grabbe, S., Sridhar, B., and Mukherjee, A., "Central East Pacific Flight Scheduling," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Hilton Head, South Carolina, August 2007.

[11]Bertsekas, D. P. and Tsitsiklis, J. N., "Some aspects of parallel and distributed iterative algorithms–a survey," *Automatica*, Vol. 27, No. 1, 1991, pp. 3–21.

[12]"Aircraft Situation Display To Industry: Functional Description and Interface Control Document," Tech. Rep. ASDI-FD-001, Volpe National Transportation Center, U.S. Department of Transportation, June 2005.

[13]Bilimoria, K., Sridhar, B., Chatterji, G., Sheth, K., and Grabbe, S., "FACET: Future ATM Concepts Evaluation Tool," *Air Traffic Control Quarterly*, Vol. 9, No. 1, 2001, pp. 1–20.

[14]Menon, P., Diaz, G., Vaddi, S., and Grabbe, S., "A Rapid-Prototyping Environment for En Route Air Traffic Management Research," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, August 2005.

[15]Makhorin, A., "GNU Linear Programming Kit, Version 4.23," http://www.gnu.org/software/glpk/glpk.html.

[16]ILOG, Inc., "ILOG CPLEX," http://www.ilog.com/products/cplex/, April 2007.

[17]Fourer, R., Gay, D., and Kernighan, B., "AMPL: A Mathematical Programming Language," 1989.

[18]Rios, J. and Ross, K., "Solving High-Fidelity, Large-Scale Traffic Flow Management Problems in Reduced Time," *AIAA Aviation Technology, Integration and Operations Conference*, Anchorage, Alaska, September 2008.